15-110: Principles of Computing, Spring 2018

Programming Assignment 2

Due: Tuesday, January 30 by 9PM on Autolab

Note: You are **responsible for protecting your solutions** to the following problems from being seen by other students both physically (e.g., by looking over your shoulder or verbal discussion) and electronically. In particular, since the lab machines use the Andrew File System (AFS) to share files worldwide, you need to be careful that you do not put files in a place that is publicly accessible.

If you are doing the assignment on the Gates-Hillman Cluster machines we use in the lab or on `unix.andrew.cmu.edu`, please remember to have your solutions inside a `private` folder (which is under your home directory). Our recommendation is that you create a `pa2` folder under `~/private/15110` for this assignment. That is, the new directory `pa2` is inside the directory named `15110`, which is inside the `private` directory.

Overview

For this assignment, you will create a Python file for each of the problems below. You should save all of these files in a folder named `pa2`. Once you have every file, you should zip up the `pa2` folder and submit the zipped file on Autolab.

As you will discover this semester, computer programs are rarely correct whey they are first written. They often have bugs. In addition to writing the code, you should test your code on multiple inputs, for which you independently know the correct output (e.g., by plugging the inputs into a calculator).

Exercises

After creating a Python file with the provided name, you will write a Python function to calculate each formula indicated below. The calculations should be performed with the full precision as shown in the examples; you do not have to round off. Note that due to floating point precision issues (to be discussed in class), your answers may differ slightly with the answers shown in the examples below.

1. [2 points] You want to determine what rate of return (i.e. annual interest rate) you need to achieve in order to start with an investment P in dollars and end up with an investment Q in dollars after n years. To compute this, you need to use the following formula:

$$i = \left(\frac{Q}{P}\right)^{1/n} - 1$$

For example, if you had $100 and you wanted a return of $200 in 5 years, you would need an annual interest rate of approximately 14.87%.

In `rate.py`, define a Python function `compute_rate(P,Q,n)` that has three parameters representing the initial investment in dollars `P`, the desired return in dollars `Q`, and the number of investment years, `n`. You may assume that `P` is always less than or equal to `Q` and that `n` is always a positive integer. Your function should calculate and **return** the interest rate needed to grow the investment from `P` to `Q` over `n` years.

Example Usage:

```
-bash-4.2$ python3 –i rate.py
>>> compute_rate(100, 200, 5)
0.1486983549970351
>>> compute_rate(15, 110, 10)
0.22047852624585262
>>> compute_rate(200, 300, 20)
0.02048015364945277
```

2. [3 points] The volume V of a tetrahedron with edge length a can be calculated using the following formula:

$$V = \frac{a^3}{6\sqrt{2}}$$

Tetrahedron: all edges have the same length *a*

(i) In `tetra.py`, define a Python function `tetra_volume(edge_length)` that has one parameter `edge_length` representing the length of the edge of the tetrahedron. This function should calculate and **return** the volume of a tetrahedron whose edge length is `edge_length`.

In order to use the built-in function for square root, you must place the following line at the beginning of your Python file:

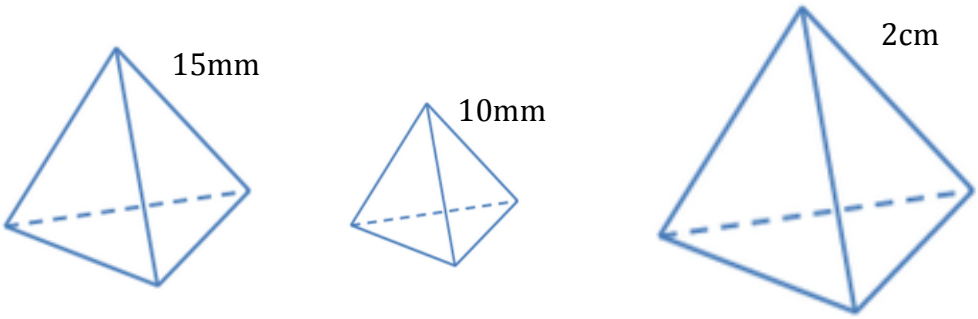
```
import math
```

Then you may use `math.sqrt` in your Python code.

Example Usage:

```
-bash-4.2$ python3 –i tetra.py
>>> tetra_volume(15)
397.74756441743295
>>> tetra_volume(110)
156859.8542932158
```

(ii) Consider the set of tetrahedrons below.



In the same file `tetra.py`, define another Python function `print_volume()` with no parameters that calls the function `tetra_volume` to calculate the volume of each of the three objects above, calculates the total volume, and then prints this value in a sentence. This function should return `None` after **printing** the following line of text:

`The total volume in cubic millimeters is XXX.XXXXXXXX`

(`XXX.XXXXXXXX` should be replaced by the calculated volume.)

Your `print_volume()` function must call the `tetra_volume` function that you wrote in part (i), instead of directly calculating the volume of the tetrahedron. Each time you call the `tetra_volume` function, store the returned answer in a variable. Once you have the three volumes in three variables, add these variables together and store the result in a fourth variable. Then print the sentence using the fourth variable for the final volume. CAREFUL: The objects above are not measured in the same units!

Example Usage:

```
-bash-4.2$ python3 –i tetra.py
>>> print_volume()
The total volume in cubic millimeters is 1458.407736197254
```

Note that Python does not display `None` even though that it is the value returned. Your final answer should be very close to the example above but might be off by a little bit due to floating point precision issues.

3. [3 points] In music, if two notes are separated by one octave, then one note is twice the frequency (pitch) than the other. Using the equal-tempered system, a musical octave can be divided into 1200 "cents". A musical note that is *x* cents above frequency *f* has a frequency of $f * 2^{x/1200}$.

   The frequency of "middle C" on a keyboard is 261.6 Hertz (Hz). The musical C one octave above middle C should be twice the frequency or 523.2 Hertz since it is 1200 cents above middle C and 261.6 Hz * $2^{1200/1200}$ = 261.6 * 2 = 523.2 Hz.

   (i) In `music.py`, define a Python function `compute_freq(freq_in_Hz, cents)` that has two parameters representing an initial frequency `freq_in_Hz` in Hertz and a number of `cents`. This function calculates and **returns** the frequency that is the given number of cents above the given frequency.

   Example Usage:

```
-bash-4.2$ python3 –i music.py
>>> compute_freq(261.6, 1200)
523.2
>>> compute_freq(261.6, 100)
277.15554548439167
```

   (ii) A major chord is made up of a root note and two notes above the root, all played together. Using equal tempered tuning, the two notes above the root in a major chord are at 400 cents and 700 cents.

   In the same file `music.py`, define a Python function `major_chord(root_freq)` that has one parameter representing the frequency of the root of the musical chord. This function prints the frequency (in Hertz) of each note of the chord. You should use the `compute_freq` function you defined above rather than compute the frequencies directly each time. This function should return `None` when it is finished.

   Example Usage:

```
-bash-4.2$ python3 –i music.py
>>> major_chord(261.6)
261.6
329.59534665249885
391.95713131093993
```

   Again, note that Python does not display `None` even though that it is the value returned, and your answers may differ slightly due to floating point precision issues.

4. [2 points] Consider a 12-hour clock that has an hour value (between 1 and 12) and a minute value (between 0 and 59). If you push the minute button m times, the minutes increase by m, wrapping around back to 0 when necessary. Pushing the minute button does not affect the hour display. For example, if the minutes display is 40 and you push the minute button 30 times, the minutes display will show 10. If the minutes display is 30 and you push the minutes button 110 times, the minutes display will show 20.

   (i) In the file `time.py`, write a Python function `add_minutes(minute, m)` that has two parameters representing the current `minute` on display and `m`, the number of times the minute button is pushed. You may assume `minute` is an integer between 0 and 59 inclusive and `m` is a positive integer. This function calculates and **returns** the new value of the minutes on display after the minute button is pushed m times.

   For this problem, do not use a loop. You should be able to compute the value to return using the modulo operator.

   Example Usage:

```
-bash-4.2$ python3 –i time.py
>>> add_minutes(20, 15)
35
>>> add_minutes(40, 30)
10
>>> add_minutes(30, 110)
20
```

   (ii) In the same file `time.py`, write a Python function `add_hours(hour, h)` that has two parameters representing the current `hour` on display and `h`, the number of times the hour button is pushed. You may assume `hour` is an integer between 1 and 12 inclusive and `h` is a positive integer. This function calculates and **returns** the new value of the hours on display after the hour button is pushed h times.

   For this problem, do not use a loop or an `if` statement. You should be able to compute the value to return using the modulo operator, although this one is a little trickier!

   Example Usage:

```
-bash-4.2$ python3 –i time.py
>>> add_hours(5, 2)
7
>>> add_hours(8, 7)
3
>>> add_hours(6, 6)
12
```

Submission

You should now have the `pa2` folder that contains the following 4 Python files:

a. `rate.py`
b. `tetra.py`
c. `music.py`
d. `time.py`

Zip up the folder and submit the zipped file named as `pa2.zip` on Autolab.

NOTE: Autolab submission will not open until Friday, January 26.